Pontifícia Universidade Católica de Minas Gerais

Curso: Desenvolvimento web Full Stack
Disciplina: Programação web com Node.js
Professor: Samuel Martins
Valor: 10 pts

Exercício 5

Passo 1

Iremos construir uma sala de bate-papo em tempo real para múltiplos! Para isso, clone o repositório a seguir e rode o comando ***npm install***: https://github.com/samwx/nodejs-socket.io.

Nesse repositório, temos como principais dependências o express, o nodemon e o socket.io.

Passo 2

Na linha de comando, rode o comando ***npm start*** e certifique de que o projeto irá iniciar corretamente

Passo 3

Com o projeto aberto no Visual Studio Code, no arquivo ***index.hbs***, crie um formulário com o id "**comment-form**" (**sem os atributos action e method**) e adicione os <inputs> "author" e "comment". **Importante**: não esqueça de colocar o atributo **name** nos inputs.

Coloque também um button com o type="submit" para submeter os dados do formulário.

Passo 4

Na linha **acima** ao formulário criado, coloque uma <ul> vazia com o id "messages", onde iremos mostrar as mensagens enviadas e recebidas:

```
<ul id="messages"></ul>
```

Passo 5

Implemente uma função no evento onsubmit do javascript (ou o $submit do jquery) que capture os dados do formulário e envie uma mensagem via websocket para o nosso servidor:

```
// Elemento do formulário
const messageForm = document.getElementById('comment-form');

// Implementação do evento onsubmit
messageForm.onsubmit = function(event) {
    // previne o recarregamento da página ao submeter o formulário
    event.preventDefault();

    const author = document.getElementById('author').value;
    const comment = document.getElementById('comment').value;
    const data = { author, comment };

    // Envia uma mensagem ao servidor com os dados recuperados
    socket.emit('sendMessage', data)
}
```

Passo 6

Implemente uma função auxiliar para renderizar um html com os dados recebidos e adicioná-lo a nossa lista de mensagens:

```
const messagesList = document.getElementById('messages')
const renderMessage = (message) => {
    const element = document.createElement('li')
    element.innerHTML = `<strong>${message.nickname}</strong>: ${message.comment}`
    messagesList.append(element)
}
```

Passo 7

Adicione o método *renderMessage* **logo antes** da linha onde enviamos uma mensagem via websocket (linha 13 do **passo 5**) para o servidor, passando como parâmetro o objeto **data**.

Passo 8

Logo abaixo da implementação do método onSubmit, ainda no index.hbs, implemente um método para receber eventos do servidor (receiver, conforme visto no slide). Escolha um nome para a mensagem e no callback, chame a função "*renderElement*" passando como parâmetro os dados recebidos (conteúdo) no callback.

Passo 9

Agora, no lado do servidor, implemente um "receiver" para a mensagem "sendMessage", e na função de callback, receba via parâmetro o objeto enviado no lado do cliente. Em seguida envie um broadcast para

todos os usuários conectados na nossa aplicação avisando que uma nova mensagem está disponível para listagem (faça isso passando na callback da função “emit” os mesmos dados recebidos no callback do evento “sendMessage”). **Importante**: para funcionar corretamente, o nome do “broadcast” enviado deve possuir o mesmo nome escolhido na mensagem do **passo 8**.

Passo 10

Para testar a solução funcionando, abra duas abas no navegador e envie uma mensagem. A mesma mensagem deve aparecer nas duas abas.